Naveen TK
naveentk1010@gmail.com
+91-9585888184

Accuknox QA Trainee Practical Assessment

Problem statement 1:

Step 1: Clone the Repository.

Via terminal:

```
git clone https://github.com/nyrahul/wisecow
cd wisecow
```

Step 2: Dockerfile creation with file name as “Dockerfile”

Dockerfile:

```
# Use an official Python runtime as a parent image

FROM python:3.9-slim

# Set the working directory in the container

WORKDIR /app

# Install git

RUN apt-get update && apt-get install -y git

# Clone the repository

RUN git clone https://github.com/nyrahul/wisecow.git

# Change the working directory to the cloned repository

WORKDIR /app

# Install any needed packages specified in requirements.txt

RUN pip freeze > requirements.txt

RUN pip install --no-cache-dir -r requirements.txt

# Expose the port the application runs on (adjust if different)

EXPOSE 5000

# Define environment variable

ENV FLASK_APP=app.py

# Run app.py when the container launches

CMD ["flask", "run", "--host=0.0.0.0", "--port=5000"]
```

Docker Image creation:

- Save the above Dockerfile content in a file named `**Dockerfile**`.
- Open a terminal, navigate to the directory containing the `Dockerfile`, and run the following command to build the Docker Image:

  ```
  docker build -t wisecow-app .
  ```

- After the image is built, need to tag the docker image using the below docker command (optional)

  ```
  docker tag wisecow-app:latest naveentk10/wisecow:latest
  ```

- Login to docker hub to push the docker image to hub using the below docker command (credentials can be passed with the command)

  ```
  docker login -u $DOCKER_USERNAME -p $DOCKER_PASSWORD
  ```

- After the image is tagged, then need to push docker image to docker hub using the below docker command

  ```
  docker push naveentk10/wisecow:latest
  ```

- After the image is built, you can run it with the following command to run the Docker Container:

  ```
  docker run -p 5000:5000 wisecow-app
  ```

Step 3: Create Kubernetes Manifest Files

Create a directory called **k8s** with the following manifest files as mentioned below:

1. Deployment Manifest (**k8s/deployment.yaml**):

```
---------------------------------------------

apiVersion: apps/v1
kind: Deployment
metadata:
 name: wisecow-deployment
spec:
 replicas: 3
 selector:
  matchLabels:
   app: wisecow
 template:
  metadata:
   labels:
    app: wisecow
```

```
  spec:
   containers:
   - name: wisecow
    image: naveentk10/wisecow:latest
```

----

https://us-east-1.console.aws.amazon.com/iam/home#/roles/details/wiscow-cluster?section=permissions – role created

Cluster deleted

2. Service Manifest (k8s/service.yaml):

```
apiVersion: v1
kind: Service
metadata:
 name: wisecow-service
spec:
 selector:
  app: wisecow
 ports:
  - protocol: TCP
   port: 5000
   targetPort: 5000
 type: LoadBalancer

 template:
  metadata:
   labels:
    app: wisecow
```

Step 4: Setup GitHub Actions for CI/CD

Need to create a “***.github/workflows”*** directory and add a **ci-cd.yaml** file:

ci-cd.yaml

```
name: CI/CD Pipeline

on:
 push:
  branches:
   - main

jobs:
 build:
  runs-on: ubuntu-latest

  steps:
  - name: Checkout repository
   uses: actions/checkout@v2

  - name: Set up Docker Buildx
   uses: docker/setup-buildx-action@v1

  - name: Login to DockerHub
   uses: docker/login-action@v1
   with:
    username: ${{ secrets.DOCKERHUB_USERNAME }}
    password: ${{ secrets.DOCKERHUB_TOKEN }}

  - name: Build and push Docker image
   uses: docker/build-push-action@v2
   with:
    push: true
    tags: your-dockerhub-username/wisecow:latest

 deploy:
  runs-on: ubuntu-latest
  needs: build

  steps:
  - name: Checkout repository
```

```
      uses: actions/checkout@v2

    - name: Configure AWS credentials
      uses: aws-actions/configure-aws-credentials@v2
      with:
        aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
        aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
        aws-region: your-aws-region


    - name: Install kubectl
      run: |
        curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
        chmod +x kubectl
        sudo mv kubectl /usr/local/bin/


    - name: Update kubeconfig for EKS cluster
      run: |
        aws eks update-kubeconfig --name wiscow --region ap-south-1

    - name: Deploy to Kubernetes
      env:
        KUBE_CONFIG_DATA: ${{ secrets.KUBE_CONFIG_DATA }}
      run: |
        kubectl --kubeconfig=kubeconfig apply -f k8s/deployment.yaml
        kubectl --kubeconfig=kubeconfig apply -f k8s/service.yaml
```

Step 5: Secure the Application with TLS

Create a Kubernetes secret to hold your TLS certificates:

```
kubectl create secret tls wisecow-tls --cert=path/to/tls.crt --key=path/to/tls.key
```

Modify the service to use the TLS secret and ingress controller:

Ingress Manifest (**k8s/ingress.yaml**):

```
-----------------------------------------------------
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: wisecow-ingress
 annotations:
  nginx.ingress.kubernetes.io/rewrite-target: /
  cert-manager.io/cluster-issuer: "letsencrypt"
spec:
 tls:
 - hosts:
  - your-domain.com
  secretName: wisecow-tls
 rules:
 - host: your-domain.com
  http:
   paths:
   - path: /
    pathType: Prefix
    backend:
     service:
      name: wisecow-service
      port:
       number: 80
-----------------------------------------------
```

Step 6: Configure Secrets in GitHub

The following secrets are added in the GitHub repository:

DOCKERHUB_USERNAME: Your DockerHub username

DOCKERHUB_TOKEN: Your DockerHub access token

Step 7: Push your changes to the repository

```
git add .
git commit -m "Initial commit with Docker and Kubernetes setup"
git push origin main
```

Verify the GitHub Actions workflow runs and deploys the application.

Problem statement 2:

1) System Health Monitoring Script:

Develop a script that monitors the health of a Linux system. It should check CPU usage, memory usage, disk space, and running processes. If any of these metrics exceed predefined thresholds (e.g., CPU usage > 80%), the script should send an alert to the console or a log file.

Python code :   “linux_health_monitor.py”

```
import psutil
import logging
import time
from datetime import datetime


# Logging configuration
logging.basicConfig(filename='LinuxHealth.log', level=logging.INFO,
              format='%(asctime)s - %(levelname)s - %(message)s')


cpu_threshold = 80
memory_threshold = 80
disk_threshold = 80
process_threshold = 200  # number of processes
```

```
def check_cpu_usage():
    cpu_usage = psutil.cpu_percent(interval=1)
    if cpu_usage > cpu_threshold:
        alert_message = f"High CPU usage detected above 80% (CPU usage : {cpu_usage}%)"
        print(alert_message)
        logging.warning(alert_message)
    return cpu_usage


def check_memory_usage():
    memory_info = psutil.virtual_memory()
    memory_usage = memory_info.percent
    if memory_usage > memory_threshold:
        alert_message = f"High memory usage detected above 80% (Memory usage : {memory_usage}%)"
        print(alert_message)
        logging.warning(alert_message)
    return memory_usage


def check_disk_space():
    disk_info = psutil.disk_usage('/')
    disk_usage = disk_info.percent
    if disk_usage > disk_threshold:
        alert_message = f"Low disk space detected: {disk_usage}% used"
        print(alert_message)
        logging.warning(alert_message)
    return disk_usage
```

```
def check_running_processes():
    processes_count = len(psutil.pids())
    if processes_count > process_threshold:
        alert_message = f"High number of running processes detected below 200 (Processes count : {processes_count})"
        print(alert_message)
        logging.warning(alert_message)
    return processes_count


def monitor_system_health():
    cpu_usage = check_cpu_usage()
    memory_usage = check_memory_usage()
    disk_usage = check_disk_space()
    processes_count = check_running_processes()

    return {
        "cpu_usage": cpu_usage,
        "memory_usage": memory_usage,
        "disk_usage": disk_usage,
        "processes_count": processes_count
    }


if __name__ == "__main__":
    while True:
        system_health = monitor_system_health()
        logging.info(f"System Health: {system_health}")
        print(f"System Health: {system_health}")
        time.sleep(120)  # Checks every 2 minutes
```

NOTE :

‘psutil’ library must be installed to run this program.

4) Application Health Checker:

Please write a script that can check the uptime of an application and determine if it is functioning correctly or not. The script must accurately assess the application's status by checking HTTP status codes. It should be able to detect if the application is 'up', meaning it is functioning correctly, or 'down', indicating that it is unavailable or not responding.

Python code: “app_health_checker.py”

```
import requests
import logging
from datetime import datetime
import time

# Logging configuration
logging.basicConfig(filename='app_health_checker.log', level=logging.INFO,
          format='%(asctime)s - %(levelname)s - %(message)s')

URL = 'http://accuknox.com/application_status/'
interval = 60  # seconds

def check_application_status(url):
    try:
        response = requests.get(url)
        if response.status_code == 200:
            logging.info(f"Application is UP (Status Code: {response.status_code})")
            print(f"{datetime.now()} - Application is UP (Status Code: {response.status_code})")
        else:
            logging.warning(f"Application is DOWN (Status Code: {response.status_code})")
            print(f"{datetime.now()} - Application is DOWN (Status Code: {response.status_code})")
    except requests.exceptions.RequestException as e:
        logging.error(f"Application is DOWN (Error: {str(e)})")
        print(f"{datetime.now()} - Application is DOWN (Error: {str(e)})")


if __name__ == "__main__":
    while True:
        check_application_status(URL)
        time.sleep(interval)
```

NOTE :

‘requests’ library must be installed to run this program